



Tecnológico de Monterrey

**Instituto Tecnológico de Estudios Superiores de
Monterrey**

TE3001B.101

Fundamentación de Robótica

Challenge 2.

Gpo 101

Profesor:

Rigoberto Cerino Jiménez

Alumnos:

Daniela Berenice Hernández de Vicente	A01735346
Alejandro Armenta Arellano	A01734879
Dana Marian Rivera Oropeza	A00830027

Fecha: 27 de Febrero del 2023

Resumen

En este reporte se podrá observar una descripción de lo que son los custom messages, parameters y launch file, cómo funcionan, todo esto con tal de comprender más allá de simplemente resolver el challenge semanal impuesto por Manchester Robotics.

Aunado a estas descripciones y definiciones, se puede encontrar la solución paso por paso al challenge de esta segunda semana.

Objetivos

En este segundo challenge se espera que los estudiantes repasen los conceptos vistos en la sesión pasada.

Por lo cual la actividad consiste en lo siguiente:

- Crear un controlador para un sistema simple de primer orden en ROS. Este sistema representa el comportamiento dinámico de un motor DC.
- El nodo Sistema y un programa de simple estructura serán proporcionados por MCR2.
- El sistema de control puede ser “P”, “PI”, o “PID”, otros controladores son aceptables sólo si el profesor lo ha autorizado previamente.

Objetivos específicos:

- Nodo System:
 - Es proporcionado por MCR2.
 - Las entradas y salidas del nodo son mensajes personalizados hechos específicamente para este sistema.
 - El estudiante deberá ocupar estos mensajes para comunicarse con el sistema.
 - Estos mensajes estarán en la carpeta “msg”.
 - Los parámetro del sistema se pueden encontrar en “config/system_params.yaml”.
- Nodo Controller:
 - Crear un nodo llamado “Controller” para generar una entrada de control al nodo system.
 - Este nodo debe publicar en el tópico “motor_input” y suscribirse a los tópicos “motor_output” y “set_point”.
 - Los mensajes del controller “motor_input” y “motor_output” son los siguientes:

```
Motor input message (/motor_input):
float32 input          #Input to the system
float64 time           #Simulation time to be sent to the system

Motor output message (/motor_output):
float32 output         #Output of the system
float64 time           #Time stamp of the system
string status          #Status of the motor "Turning", "Not Turning", "Max Speed"
```

- La salida del controller “motor_input” debe ser delimitado entre el intervalo -1 a 1 i.e., $u(k) \in [-1, 1]$.
- El mensaje para el tópico “set_point” debe ser definido por el estudiante.
 - Debe ser un mensaje personalizado.

- Debe incluir al menos dos diferentes variables.
- El nodo control, debe usar un archivo parámetro, para todas las variables de ajuste requeridas.
- El tiempo de muestreo y la velocidad pueden ser los mismos que el nodo system.
- Queda estrictamente prohibido utilizar alguna otra librería para Python además de NumPy. El controlador debe hacerse sin usar ningún controlador en línea predefinido.
- Launch File and Plotting:
 - Usar la herramienta “rqt_plot” para imprimir ambas señales.
 - Crear un archivo launch el cual debe poder acceder a las terminales requeridas para imprimir la información.
 - Debe ser capaz de abrir el “rqt_plot” e imprimir todas las señales requeridas en una misma ventana.
 - Debe ser capaz de cargar todos los archivos de parámetros automáticamente.

Introducción

Semanalmente se nos presentará un reto por Manchester Robotics, los cuales serán resueltos con la ayuda de programas especializados como ROS y ciertos lenguajes de programación como Python.

De igual manera cada semana contaremos con una clase base con respecto a lo que se manejara durante el challenge semanal, así como una sesión de preguntas y respuestas donde se buscarán resolver dudas con respecto al mismo challenge de la semana.

Por otro lado también se nos proporcionarán algunas actividades las cuales hasta cierto punto serán un pequeño desglose del challenge semanal, claro está que contarán con una menor dificultad.

En esta segunda semana al ya contar con una noción clara de la arquitectura de trabajo de ROS, se nos introdujo a los siguientes temas:

- ROS Namespaces:
 - Un namespace en ROS se puede visualizar como un directorio que contiene artículos con diferentes nombres.
 - Esos artículos pueden ser nodos, tópicos, o otros namespaces (dependiendo de la jerarquía).
 - Existen varias formas de definir los namespaces. La más fácil es por una línea de comando, la cual es muy sencilla, pero para proyectos largos no es recomendada.
- ROS Parameters:
 - Son variables con algunos valores predefinidos que se almacenan en archivos separados o codificados en algún programa tal que el usuario pueda acceder fácilmente para modificar el valor.
 - Al mismo tiempo, los parámetros se pueden compartir entre diferentes programas para evitar reescribirlos o recompilar los nodos (C++).
 - ROS usa parámetros para evitar crear dependencias o reescribir nodos.

- ROS utiliza este “diccionario” para almacenar y compartir los parámetros para ser usados por sus nodos. Este “diccionario” es llamado “Parameter Server”.
- ROS Parameter Server:
 - Como se mencionó anteriormente ROS permite cargar variables en un servidor, ejecutado por el master el cual es accesible por todos los nodos del sistema.
 - Los nodos utilizan este servidor para almacenar y recuperar los parámetros al momento de ser ejecutados.
 - Esta es una librería estática vista globalmente.
 - Los parámetros están compuestos por un nombre y un tipo de dato, ROS puede usar los siguientes tipos de parámetros:
 - 32-bit integers.
 - Strings.
 - base64-encoded binary data.
 - Booleans.
 - iso8601 dates
 - Doubles.
 - Lists.
 - Los parámetros pueden ser globales con un namespace local o privados, específicos para un nodo.
 - Es útil señalar que el servidor de parámetros depende de la ejecución del master ROS no al tiempo de ejecución del nodo. Por lo tanto, si se mata un nodo pero no al master, el parámetro mantendrá el valor si fue modificado.
- ROS Parameter Files:
 - ROS ofrece la capacidad de definir parámetros mediante un parameter file.
 - Este tipo de archivos son archivos de configuración, escritos en YAML. Estos archivos son usados comúnmente en otros lenguajes para configurar parámetros o variables.
- ROS Custom Messages:
 - ROS tiene algunos mensajes predefinidos como `std_messages`, `geometric_messages`, etc.
 - A veces, se requiere que las estructuras del mensaje sean modificadas para una aplicación personalizada.
 - ROS permite al usuario personalizar mensajes y crear nuevos mensajes.
 - Los custom messages son una manera de personalizar tus propios mensajes para un motivo o aplicación en específico.
 - Los custom messages son creados por el usuario y deben estar vinculados al paquete donde se usarán.

Solución del problema

Los nodos algunos han sido creados y otros los ha proporcionado MCR2, de igual manera el controlador ha sido creado y obtenido mediante prueba y error.

Por otro lado, los comandos principales para el código han sido vistos en el challenge de la semana pasada, por lo cual en este los hemos omitido.

Aunado a esto, describimos los códigos implementados en la solución del challenge semanal.

Nodo Controller:

```
1  #!/usr/bin/env python
2  import rospy
3  import numpy as np
4  from std_msgs.msg import Float32
5  from pid_control.msg import set_point
6  from pid_control.msg import motor_input
7  from pid_control.msg import motor_output
8
9  kp = rospy.get_param("kp",5)
10 ki = rospy.get_param("ki",0.1)
11 tau = rospy.get_param("tau",6.28)
12 R = rospy.get_param("R",0.01)
13 dt = rospy.get_param("dt",5)
14
15 signal_data = 0.0
16 time_data = 0.0
17
18 output_data = 0.0
19 time_data2 = 0.0
20
21 error = 0.0
22 angularVelocity = 0.0
23
24 error_acumulado = 0.0
25 ultima_medicion = 0.0
26
27
28 msg = motor_input()
29 msg.time = 0.0
30 msg.input = 0.0
31
32
33 def callback(msg):
34     global signal_data, time_data
35     signal_data = msg.signal_y
36     time_data = msg.time_x
37
38 def callback2(msg):
39     global output_data, time_data2
40     output_data = msg.output
41     time_data2 = msg.time
42
43 def stop():
44     #Setup the stop message (can be the same as the control message)
45     print("Stopping")
46
47
48 if __name__=='__main__':
49     #Initialise and Setup node
50     pub=rospy.Publisher("motor_input",motor_input, queue_size=10)
51     pub_controlador=rospy.Publisher("control",Float32, queue_size=10)
52     rospy.init_node("controller")
53     rospy.Subscriber(rospy.get_namespace()+"set_point", set_point, callback)
54     rospy.Subscriber(rospy.get_namespace()+"motor_output", motor_output, callback2)
55     rate = rospy.Rate(10)
56     rospy.on_shutdown(stop)
```

```

58     while not rospy.is_shutdown():
59
60         error = 0.0 - signal_data
61         error_acumulado += error * dt ## multiplicar por "dt"
62         accion_proporcional = kp * error
63         accion_integral = ki * error_acumulado
64         accion_control = accion_proporcional + accion_integral
65         u = accion_control * R
66         velocidad = ultima_medicion + ((u - ultima_medicion) / tau)
67         #pub.publish(Float32(velocidad))
68         ultima_medicion = velocidad
69         signal=ultima_medicion
70
71         msg.time = time_data
72         msg.input = signal
73         pub.publish(msg)
74         pub_controlador.publish(signal)
75         print_info = "%3f | %3f" %(signal,time_data)
76         rospy.loginfo(print_info)
77         rate.sleep()

```

En este código se crea un control en la entrada del nodo sistema. De igual manera el nodo publica en el tópico “motor_input” y se suscribe a los tópicos “motor_output” y “set_point”. La salida del controller “motor_input” está delimitado entre el intervalo -1 a 1 i.e., $u(k) \in [-1,1]$ y se mandan los mensajes personalizados por nosotros.

Nodo Set Point Generator:

```
1  #!/usr/bin/env python
2  import rospy
3  import numpy as np
4  from pid_control.msg import set_point
5  from std_msgs.msg import Float32
6
7  msg = set_point()
8  msg.time_x = 0.0
9  msg.signal_y = 0.0
10
11 if __name__ == '__main__':
12     pub_signal=rospy.Publisher("set_point",set_point, queue_size=10)
13     pub_motor=rospy.Publisher("signal",Float32, queue_size=10)
14     rospy.init_node("Set_Point_Generator")
15     rate = rospy.Rate(10)
16     t=0
17
18     while not rospy.is_shutdown():
19         t+= 0.1
20         signal= np.sin(t)
21         signal=((t * np.cos(5)) + np.cos(t) * np.sin(5))* -np.cos(t)
22
23         msg.time_x = t
24         msg.signal_y = signal
25         pub_signal.publish(msg)
26         pub_motor.publish(signal)
27         print_info = "%3f | %3f" %(signal,t)
28         rospy.loginfo(print_info)
29
30         rate.sleep()
```

En este código se utiliza el código creado en el challenge 1, el cual publica en el tópico “Set_point” una señal y un mensaje.

Nodo System:

```

1  #!/usr/bin/env python
2  import rospy
3  import numpy as np
4  from std_msgs.msg import Float32
5  #from pid_control.msg import set_point
6  from pid_control.msg import motor_input
7  from pid_control.msg import motor_output
8
9  signal_data2 = 0.0
10 time_data2 = 0.0
11
12 output = motor_output()
13 output.time = 0.0
14 output.output = 0.0
15
16 #Setup parameters, variables and callback functions here (if required)
17 def callback(msg):
18     global signal_data2, time_data2
19     signal_data2 = msg.input
20     time_data2 = msg.time
21
22 #Stop Condition
23 def stop():
24     #Setup the stop message (can be the same as the control message)
25     print("Stopping")
26
27 if __name__=='__main__':
28     #Initialise and Setup node
29     rospy.init_node("system")
30     rospy.Subscriber(rospy.get_namespace()+"motor_input", motor_input, callback)
31     rate = rospy.Rate(10)
32
33     pub=rospy.Publisher("motor_output",motor_output, queue_size=10)
34     rospy.on_shutdown(stop)
35
36     #Setup Publishers and subscribers here
37     print("The Controller is Running")
38     #Run the node
39     while not rospy.is_shutdown():
40         #Write your code here
41         signal = signal_data2
42         #rospy.loginfo("The proessed signal value is: %f at a time %f",signal, time_data)
43         #pub.publish(signal_data)
44
45         output.time = time_data2
46         output.output = signal
47         pub.publish(output)
48         #rospy.loginfo("The generated signal value is: %f at a time %f", signal, time_data2)
49         print_info = "%3f | %3f" %(signal,time_data2)
50         rospy.loginfo(print_info)
51
52     rate.sleep()

```

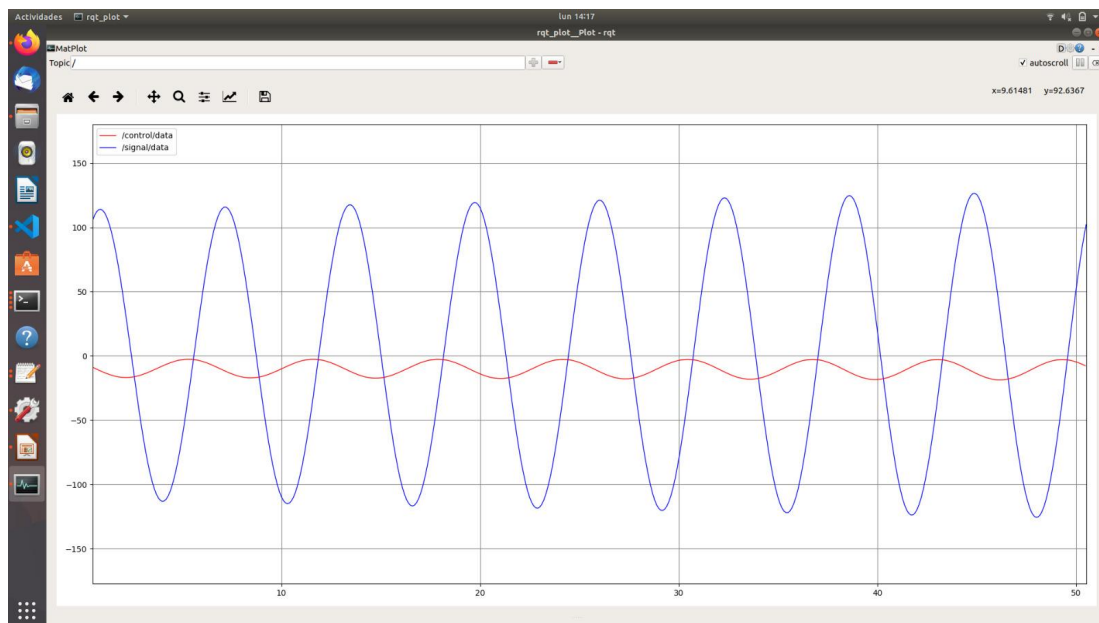
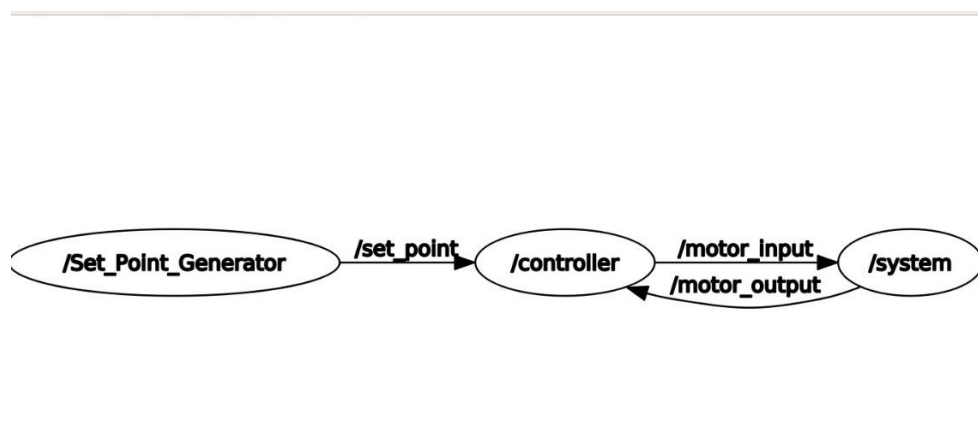
En este código se utiliza el node proporcionado por MCR2 que simula el sistema de primer orden $Y(s)/U(s) = K/(Ts+1)$ aproximando de esta forma un simple motor DC con su encoder y driver.

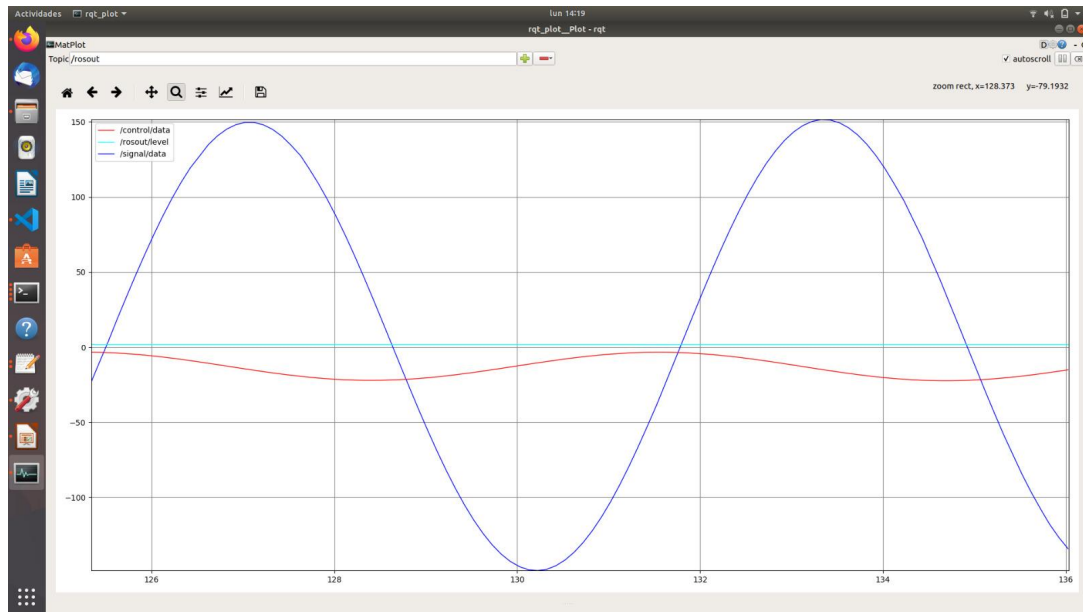
La entrada y salida del nodo son mensajes personalizados que han sido creados específicamente para el sistema.

Resultados

En este apartado se muestra la relación que existe entre los canales de comunicación y entre los nodos para la transmisión de la señal de entrada y salida, y su procesamiento.

Todo esto gracias a que el nodo `set_point_generator` manda la información a través de la señal `set_point` el cual llega al segundo nodo “controller” donde la información se ve modificada por el controlador previamente establecido, posteriormente se manda la información mediante la señal llamada “motor_input”, la cual llega al tercer nodo “system”, siendo este el que regrese la información hacia el nodo “controller” mediante la señal “motor_output”, todo esto con tal de que la información pueda seguir siendo modificada y así generar un ciclo.





Conclusiones

Aún cuando en algunos momentos parecía inalcanzable, se consiguió realizar con éxito cada uno de los objetivos antes mencionados.

Se logró obtener la creación de un controlador para un sistema simple de primer orden en ROS, que representará el comportamiento dinámico de un motor DC.

Se logró obtener el diagrama y la señal esperada por MCR2.